1. Introduction to Programming
    1. Systems Development Life Cycle
    2. Bloodshed Dev-C++ 5 Compiler/IDE
    3. Modularization and C++ Program Layout

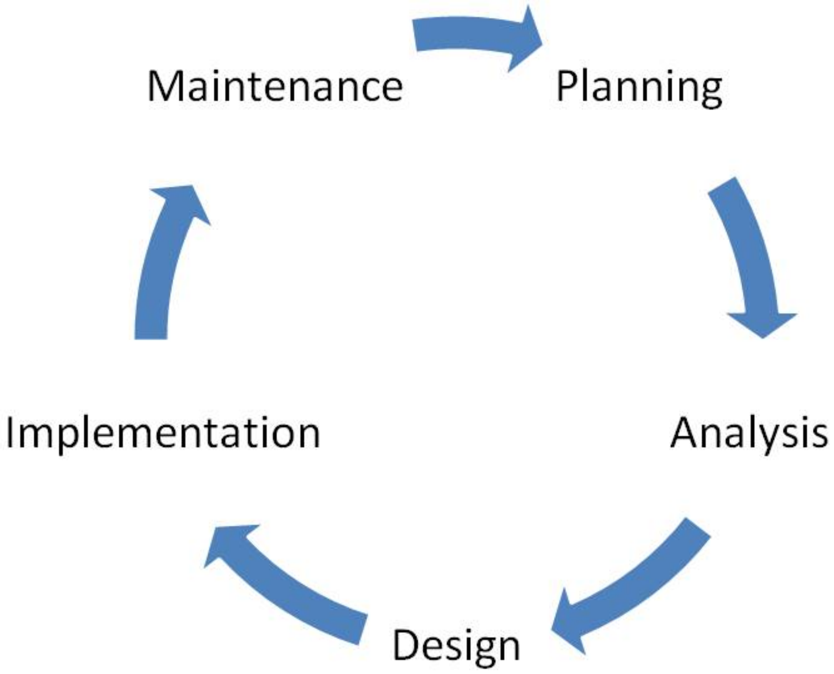
Systems Development Life Cycle
An overview of the Systems Development Life Cycle.

Discussion

The **Systems Development Life Cycle** is the big picture of creating an information system that handles a major task (referred to as an application). The **applications** usually consist of many programs. An example would be the Department of Defense supply system, the customer system used at your local bank, the repair parts inventory system used by car dealerships. There are thousands of applications that use an information system created just to help solve a business problem.

Another example of an information system would be the "101 Computer Games" software you might buy at any of several retail stores. This is an entertainment application, that is we are applying the computer to do a task (entertain you). The software actually consists of many different programs (checkers, chess, tic tac toe, etc.) that were most likely written by several different programmers.

Computer professionals that are in charge of creating applications often have the job title of **System Analyst**. The major steps in creating an application include the following and start at **Planning** step.



Systems Development Life Cycle

During the **Design** phase the System Analyst will document the inputs, processing and outputs of each program within the application. During the **Implementation** phase programmers would be assigned to write the specific programs using a programming language decided by the System Analyst. Once the system of programs is tested the new application is installed for people to use. As time goes by, things change and a specific part or program might need repair. During the **Maintenance** phase, it goes through a mini planning, analysis, design and implementation. The programs that need modification are identified and programmers change or repair those programs. After several years of use, the system usually becomes obsolete. At this point a major revision of the application is done. Thus the cycle repeats itself.

Definitions

system analyst
Computer professional in charge of creating applications.

applications
An information system or collection of programs that handles a major task.

life cycle
Systems Development Life Cycle: Planning - Analysis - Design - Implementation - Maintenance

implementation
The phase of a Systems Development Life Cycle where the programmers would be assigned to write specific programs.

Bloodshed Dev-C++ 5 Compiler/IDE
An introduction to the Bloodshed Dev-C++ 5 compiler/IDE (Integrated Development Environment) listing the advantages of using an open source compiler that works on a flash drive. The software along with installation instructions and a test program are included.

Introduction

Microsoft and Borland are the two reputable names within the programming world for compilers. They sell compiler software for many programming languages. For the C++ programming language, the Microsoft Visual Studio which includes C++ and Borland C++ Builder are excellent compilers. Often with textbooks or free via the internet; you can get Microsoft's Visual C++ Express or Borland's Personal Edition version of a compiler. However, installing either of these compliers can be complex. Microsoft's Visual Studio compiler often creates a variety of installation problems (such as making sure the operating system and .net components are current) thus making it difficult for students to install at home. These compliers require you to build a project to encompass every program. Using a commercially sold compiler that professional programmers would consider using for project development is fine for professionals but often confusing to beginners. Eventually, if you are going to become a professional programmer, you will need to become familiar with the commercially sold compilers.

We suggest that beginning students consider one of the easier to install compiler software packages for use in a programming fundamentals course. The best option we have found is an **open source** compiler/IDE (Integrated Development Environment) named: **Bloodshed Dev-C++ 5 compiler/IDE**.

open source

> Group development of source code for software that is made available to the public at no cost.

Bloodshed Dev-C++ 5 compiler/IDE

Advantages: Can be installed on Windows 95/98/NT/2000/XP operating systems. I have it installed on Windows Vista operating system, thus it can work with slower processors and almost any Windows operating system. It only requires about 80 MB of storage space (usually enough for the compiler with all of its files and storage room for several of your programs). It is very easy to install and easy to use. Does not require the use of a "project"; thus individual source code files can be easily compiled.

Disadvantages: Would not normally be used by professional programmers, but is sufficient for a beginning computer programming course and is a full-featured compiler/IDE.

Unique Advantage: Can be installed and run on a flash drive, thus giving the student the ability to work on their lab assignments on any computer that has a USB port. This can give the student **portability**, being able to do lab assignments at home, work, library, open lab, classroom, friend’s house, etc.

portability

> The ability to transport software on a flash drive and thus use it on various machines.

Preparation before Installation

Creating the Needed Folders and Sub-Folders

You need to get the **software** and a C++ **source code** program that has been tested and is error free. You will need about 80MB of storage space. We suggest that you create **two folders** on your hard drive or flash drive depending on which installation you choose. If on a flash drive create them at the root level of the drive. If on your home machine, you can use the folder area set up by the operating system for you as a user. Name them:

- Cpp_Software_Download
- Cpp_Source_Code_Files

Within the Cpp_Source_Code_Files folder, create a sub-folder named:

- Compiler_Test

To help you keep files organized, you will want to create other sub-folders for storing source code files. We suggest you create at least two other sub-folder to be used with Connexions' related modules. Within the Cpp_Source_Code_Files, create sub-folders named:

- Demo_Programs
- Monitor_Header

folder
: A named area for storage of documents or other files on a disk drive or flash drive.

source code
: Any collection of statements or declarations written in some human-readable computer programming language.

Getting the Software

The full version of the software is named: Dev-C++ 5.0 beta 9.2 (4.9.9.2) (9.0 MB) with Mingw/GCC 3.4.2 You can either download it from Bloodshed or download the version as of 12/8/2008 that is stored on the Connexions web site. Store it in the Cpp_Software_Download folder you created. The software is approximately 9.1 MB and will take several minutes to download if you are using a dial-up modem connection.

> **Note:**The software has not significantly changed since 2007 and the Connexions version will be sufficient for most users. The Bloodshed link requires some additional navigation to get to the software download. Thus, because it is significantly easier, we recommend that you download the software from the Connections web site.

Link to Bloodshed: http://www.bloodshed.net/dev/devcpp.html

Download from Connexions: devcpp-4.9.9.2_setup.exe

Getting a C++ Source Code File

Listed below is a C++ source code file titled: Compiler_Test.cpp It has been prepared for Connexions web delivery. Download and store it in the Compiler_Test sub-folder you created. You may need to right click on the link and select "Save Target As" in order to download the file.

Download from Connexions: Compiler_Test.cpp

Installation Instructions for Bloodshed Dev-C++ 5 compiler/IDE

The Version 5 which is well tested (don’t let the beta release scare you) and should work on a variety of machines and various Microsoft Operating systems including Windows 98, Windows 2000, Windows XP and Windows Vista. Below are installation instructions for installing it on a machine or installing it on a flash drive. We don’t suggest trying to switch between the machine drive and flash drive. If it is installed on the machine drive and you try installing it on a flash drive, it creates problems and will not work perperly. Either install it on the flash drive to gain your portability or install it on your machine.

Computer Installation Instructions

1. Navigate to the Cpp_Software_Download folder and run the **devcpp-4.9.9.2_setup.exe** software by double cliking on the filename.
2. Use common sense and answer the installation prompts. NOTE THE FOLLOWING TWO ITEMS:
3. When it gets to the "Choose Install Location" use the default software location of: **C:\Dev-Cpp\** (or select the location you want to store the

installed program but use the default unless you are familiar with installing software).

4. When it asks: "Do you want to install Dev C++ for all users on this computer?" answer "Yes".
5. After it installs, it will ask some "first time configuration" questions. Again, use common sense and answer the questions. NOTE THE FOLLOWING ITEM:
6. Answer "No" to the retrieve information from header files.
7. It will start your compiler/IDE with a "Tip of the day". We suggest you check the box in the lower left and select "Close".
8. Close your compiler/IDE by using the normal red "X" box. We want to show you how to start your compiller normally.
9. You start your compiler software similar to starting any software loaded on your machine ("Start" then "All Programs" then "Bloodshed Dev-C++" then "Dev-C++").
10. On the menus at the top – Select "File" then "Open project or file" then navigate to where your source code file (Compiler_Test.cpp) is stored on your hard drive. See the suggested folder name above. Select the source code file and open it.
11. You should see the source code listing. Press F9 key or select the "Execute" then "Compile & Run" from the menus at the top. A black screen box should appear and you answer questions appropriately to run the program. When you are done running your program the black screen box goes away.

Flash Drive Installation Instructions

1. Navigate to the Cpp_Software_Download folder and run the **devcpp-4.9.9.2_setup.exe** software by double cliking on the filename.
2. Use common sense and answer the installation prompts. NOTE THE FOLLOWING TWO ITEMS:
3. When it gets to the "Choose Install Location" you can see that the default software location of: **C:\Dev-Cpp\** however, it needs to be changed. Change the "Destination Directory" by selecting changing the default software location from: **C:\Dev-Cpp\** to **DriveLetter:\Dev-**

Cpp\ (where the **DriveLetter** is the drive that represents your flash drive).

4. When it asks: "Do you want to install Dev C++ for all users on this computer?" answer "No".
5. After it installs, it will ask some "first time configuration" questions. Again, use common sense and answer the questions. NOTE THE FOLLOWING ITEM:
6. Answer "No" to the retrieve information from header files.
7. It will start your compiler/IDE with a "Tip of the day". We suggest you check the box in the lower left and select "Close".
8. Close your compiler/IDE by using the normal red "X" box. We want to show you how to start your compiller normally.
9. To start your compiler software you navigate to the "Dev-Cpp" folder on your flash drive and select the "devcpp.exe" application. NOTE: When using the flash drive you should not try starting the compiler by double clicking on a C++ source code file. This method works on a machine installation but does not work on a flash drive installation.
10. On the menus at the top – Select "File" then "Open project or file" then navigate to where your source code file (Compiler_Test.cpp) is stored on your flash drive. See the suggested folder name above. Select the source code file and open it.
11. You should see the source code listing. Press F9 key or select the "Execute" then "Compile & Run" from the menus at the top. A black screen box should appear and you answer questions appropriately to run the program. When you are done running your program the black screen box goes away.

Modularization and C++ Program Layout
An introduction of the concepts of modular programming with a simple example of program control and specific task functions using a C++ source code listing. The general layout of a C++ program is described.

Concept of Modularization

One of the most important concepts of programming is the ability to group some lines of code into a unit that can be included in our program. The original wording for this was a sub-program. Other names include: macro, sub-routine, procedure, module and function. We are going to use the term **function** for that is what they are called in the two predominant programming languages of today: C++ and Java. Functions are important because they allow us to take large complicated programs and to divide them into smaller manageable pieces. Because the function is a smaller piece of the overall program, we can concentrate on what we want it to do and test it to make sure it works properly. Generally functions fall into two categories:

1. **Program Control** - Functions used to simply sub divide and control the program. These functions are unique to the program being written. Other programs may use similar functions maybe even functions with the same name, but the content of the functions are almost always very different.
2. **Specific Task** - Functions designed to be used with several programs. These functions perform a specific task and thus are useable in many different programs because the other programs also need to do the specific task. Specific task functions are sometimes referred to as building blocks. Because they are already coded and tested, we can use them with confidence to more efficiently write a large program.

The main program must establish the existence of functions used in that program. Depending on the programming language, there is a formal way to:

1. define a function (it's **definition** or the code it will execute)
2. **call** a function

3. declare a function (a **prototype** is a declaration to a complier)

Program Control functions normally do not communicate information to each other but use a common area for variable storage. Specific Task functions are constructed so that data can be communicated between the calling program piece (which is usually another function) and the function being called. This ability to communicate data is what allows us to build a specific task function that may be used in many programs. The rules for how the data is communicated in and out of a function vary greatly by programming language, but the concept is the same. The data items passed (or communicated) are called parameters. Thus the wording: **parameter passing**. The four data communication options include:

1. no communication in with no communication out
2. some communication in with no communication out
3. some communication in with some communication out
4. no communication in with some communication out

Introduction of Functions within C++

We are going to consider a simple program that might be used for testing a compiler to make sure that it is installed correctly.

Example:
Compiler_Test.cpp source code

```
//******************************************************
// Filename: Compiler_Test.cpp
// Purpose:  Average the ages of two people
// Author:   Ken Busbee; © Kenneth Leroy Busbee
// Date:     Jan 5, 2009
// Comment:  Main idea is to be able to
//           debug and run a program on your
```

```
compiler.
//************************************************
******

// Headers and Other Technical Items

#include <iostream>
using namespace std;

// Function Prototypes

void pause(void);

// Variables

int     age1;
int     age2;
double  answer;

//************************************************
******
// main
//************************************************
******

int main(void)
  {
  // Input
  cout << "\nEnter the age of the first person ---
>: ";
  cin >> age1;
  cout << "\nEnter the age of the second person --
>: ";
  cin >> age2;

  // Process
  answer = (age1 + age2) / 2.0;
  // Output
  cout << "\nThe average of their ages is --------
>: ";
  cout << answer;

  pause();
  return 0;
  }

//************************************************
******
// pause
//************************************************
******

void pause(void)
  {
  cout << "\n\n";
  system("PAUSE");
  cout << "\n\n";
  return;
  }

//************************************************
******
// End of Program
//************************************************
******
```

This program has two functions, one from each of our categories. The technical layout of functions are the same, it is our distinction that creates the two categories based on how a function is being implemented.

Program Control Function

The main program piece in C++ program is a special function with the **identifier name** of main. The special or uniqueness of main as a function is that this is where the program starts executing code and this is where it usually stops executing code. It is usually the first function defined in a program and appears after the area used for includes, other technical items, declaration of prototypes, the listing of global constants and variables and any other items generally needed by the program. The code to define the function main is provided; however, it is not prototyped or usually called like other functions within a program. In this simple example, there are no other program control functions.

Specific Task Function

We often have the need to perform a specific task that might be used in many programs. In the Compile_Test.cpp source code above we have such a task that is used to stop the execution of the code until the user hits the enter key. The functions name is: pause. This function is not communicating any information between the calling function and itself, thus the use of the data type void.

Example:
general layout of a function

```
<return value data type> function identifier
name(<data type> <identifier name for input
value>)
  {
  lines of code;

  return <value>;
  }
```

There is no semi-colon after the first line. Semi-colons are used at the end of a statement in C++, but not on the first line when defining a function. Functions have a set of **braces** {} used for identifying a group or block of statements or lines of code. There are normally several lines of code within a function. Lines of code containing the instructions end in a semi-colon. Can you identify the definition of the pause function in the above program example? The pause function definition is after the function main. Though not technically required, most programs list all functions (program control or specific task) after the function main.

Let's identify the location where the function pause is called. The calling function is the function main and it towards the end of the function. The line looks like:

```
pause();
```

When you call a function you use its identifier name and a set of parentheses. You place any data items you are passing inside the parentheses, and in our example there are none. A semi-colon ends the statement or line of code. After our program is compiled and running, the lines of code in the function main are executed and when it gets to the calling of the pause function, the control of the program moves to the pause function and starts executing the lines of code in the pause function. When it’s done with the lines of code, it will return to the place in the program that called it (in our example the function main) and continue with the code in that function.

Once we know how to define a function and how to call a function, we usually will need to know how to declare a function to the compiler (called a prototype). Because of normal computer programming industry standards, programmers usually list the function main first with other functions defined after it. Then somewhere in the function main, we will call a function. When we convert our source code program to an executable version for running on our computer, the first step of the process is compiling. The compiler program demands to know what the communication will be between two functions when a function is called. It will know the communication (what going in and out as parameters) if the function being called has been defined. But, we have not defined that function yet; it is defined after the function main. To solve this problem, we show the compiler a prototype of what the function will look like (at least the communication features of the function) when we define it.

`void pause(void);`

This line of code looks exactly like the first line in our function definition with one important addition of a semi-colon. Prototypes (or declarations to the compiler of the communications of a function not yet defined) are placed near the top of the program before the function main. **Summary concept: If you call a function before it is defined you must prototype it before it is called.** Looking at our list of the three things you do in conjunction with a function in the order that they normally appear in a program, there is a formal way to:

1. declare a function (a prototype is a communications declaration to a complier)
2. call a function
3. define a function

C++ Program Layout

From the above example, you can see that 2/3 of the program is the two functions. Most C++ programs have several items before the function main. As in the example, they often are:

1. Documentation – Most programs have a comment area at the start of the program with a variety of comments pertinent to the program. Any line starting with two slashes // is a comment and the compiler software disregards everything from the // to the end of the line.
2. #include<iostream> – This line of code inserts a file into the source code. The file contains necessary code to be able to do simple input and output.

3. using namespace std – The C++ compiler has an area where it keeps the identifier names used in a program organized and it is called a namespace. There is a namespace created in conjunction with the iostream file called: std. This line informs the compiler to use the namespace std where the identifier names in the iostream are established.
4. Function prototypes have already been explained.
5. We need some variables (storage areas) for this program to work. They are defined next.

Definitions

modularization

The ability to group some lines of code into a unit that can be included in our program.

function

What modules are called in the two predominant programming languages of today: C++ and Java.

program control

Functions used to simply sub divide and control the program.

specific task

Functions designed to be used with several programs.

parameter passing

How the data is communicated in to and out of a function.

identifier name

The name given by the programmer to identify a function or other program items such as variables.

function prototype

A function's communications declaration to a complier.

function call

A function's using or invoking of another function.

function definition
The code that defines what a function does.

braces
Used to identify a block of code in C++.